Supporting System Development by Novice Software Engineers Using a Tutor-Based Software Visualization (TubVis) Approach

Shahida Sulaiman, NurAini Abdul Rashid, Rosni Abdullah
School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia.
{shahida, nuraini, rosni}@cs.usm.my

Sarina Sulaiman
Faculty of Computer Science & Information System, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia.
sarina@ utm.my

Abstract

Most computer-aided software engineering (CASE) products provide visualization utility to view software artefacts developed. Nevertheless, existing methods or approaches in such tools are limited to generating the views or component dependencies that is focusing on 'what' the output of reverse engineering process. The online help provided by the tools only indicate 'how' to use the tools to generate the views. Since existing tools mostly target for experienced software engineers, they tend to overlook the need of explaining 'why' the output is recommended or not with regard to software engineering principles. Hence we propose tutor-based software visualization (TubVis) approach in SoVis tool that analyses software artefacts pertaining to software engineering best practices inputted by the experts and generate a set of recommendations regarding the design and coding for a novices. We anticipate TubVis can improve the quality of software design and program comprehension by combining practical and theoretical aspects of software engineering education in a software visualization tool.

1. Introduction

Computer-aided software engineering (CASE) products normally comprise visualization tools. Either research prototypes such as Rigi [15] or commercial tools such as Rational Rose [14] have the objectives to support system development activities. Besides, the majority of existing Integrated Development Environments (IDEs) for instance Borland JBuilder [2] provide visualization utility to view software artefacts that are being developed. Reverse engineering is required to visualise software artefacts in such tools.

978-1-4244-2328-6/08/$25.00©2008 IEEE

Reverse engineering is the process of analysing a subject system to identify the system's components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction [5]. By having a software visualization tool in a reverse engineering environment, extracted software artefacts and their interrelationships can be visualised to aid software engineers' program comprehension.

Existing tools employ various methods and approaches [3][9][19] for software visualization with the main goal to improve program comprehension of written software systems. However existing methods or approaches are limited to generating the views or component dependencies that is focusing on 'what' the output of reverse engineering process. The online help provided by the tools only indicate 'how' to use the tools to generate the views. Since existing tools mostly target for experienced software engineers, they tend to overlook the need of explaining 'why' the output is recommended or not recommended.

Computer science students or novice software engineers must be guided whether programs they have written are well designed or not. Existing tools provide the automation of software visualization in practical but they are lack of theoretical aspects. Typically, computer science students learn theories of software engineering during their studies. By integrating theoretical aspects in a tutor-based approach of software visualization, the students will be able to balance practical and theoretical aspects during software development and maintenance. This will make the tool more beneficial and vital in giving them theoretical guidance even to novice software engineers. However it is crucial to highlight that experienced or expert software engineers may find this

approach relatively useful to ensure their software designs are conformed to best practices. The proposed work focuses on checking the design of written source codes in a reverse engineering environment.

Novice software engineers described in this paper refer to both computer science students and software practitioners who develop software systems but do not fully practise software engineering discipline. No specific definition given by existing work because most of them refer their subjects as either computer science students or software engineers. However in our study, novice software engineers do not only refer to computer science students but also practitioners who do not fully adhere to software engineering practices and disciplines while developing software.

In the following sections we will discuss the motivation, the proposed tutor-based visualization (TubVis) approach, the example of TubVis implementation in a prototype tool (SoVis), related work and conclusion.

2. The motivation

Based on the observations and evaluations of reports produced by undergraduates' and post graduates' software development projects, the problems faced by computer science students or novice software engineers including how to create a good requirement analysis, how to transform the requirement into a proper design using certain modelling notation such as Unified Modelling Language (UML), how to convert the design into source codes and how to relate the diagrams produced during different stages of software development or maintenance with the source codes.

The majority of students make common mistakes in modelling the database of their projects. For instance in Figure 1, the class diagram created promotes circular design that will later be implemented while writing the source codes. In most cases, they do not convert the design class diagram into the source codes correspondingly even though they implement an object-oriented programming.

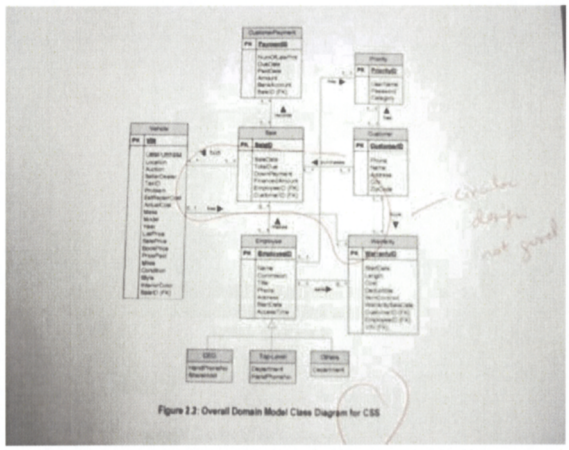

Figure 1: An example of a student's class diagram using UML notation that promotes circular design

In addition, they can hardly understand the smooth transition among different diagrams generated. Hence the diagrams produced mostly do not correspond with what they code during implementation phase. This problem has been scrutinised by some work focusing on the ability of students or novice designers to understand object-oriented software [6][8][16][17], comprehend programs using animation [11][12] and understand software for maintenance [1][18][23].

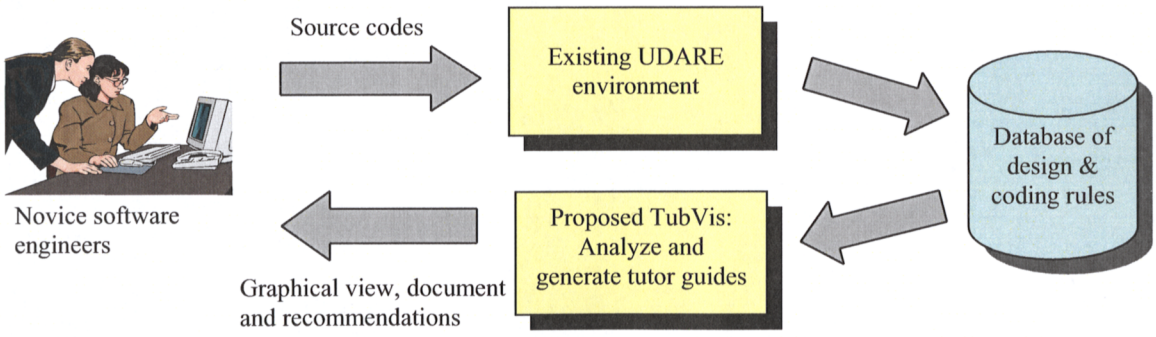

Figure 2: The proposed tutor-based approach for software visualization in UDARE environment

978-1-4244-2328-6/08/$25.00©2008 IEEE

Based on the literature study conducted so far, none of existing work attempts to provide a tutor-based approach as the guidance for the novice while developing software system through out the phases. Hence this has motivated us to focus on how to improve novices' understanding particularly in design and coding phases by integrating both theoretical and practical aspects of software engineering education.

Budgen [4] highlights how design quality can be assessed based on quality concepts, design attributes, and counts. We will focus on design attributes that include simplicity, modularity, and information hiding. In this work, we emphasize the modularity attribute that covers coupling (a measure of inter-module connectivity), and cohesion (a measure of 'functionally related' components in a module). In addition, our work also considers circularity as an attribute to be avoided in order to produce a good design.

3. Tutor-based visualization (TubVis) approach

This paper improves our earlier work [22] that proposes a tutor-based approach (TubVis) for software visualization tool to support program comprehension and software design by novice software engineers. The variables involved include software engineers or computer science students who develop a software system, the software system to be reverse engineered, the extracted software artefacts, rules of best practices in software engineering, recommendations produced and program comprehension. The attributes to be considered include the novice's level of expertise, the size of software systems, and the quality of software designed and developed.

The main research question is: *How to produce a software visualization tool that can provide both practical and theoretical guidance in software engineering while designing and coding software systems?* The null hypothesis to be rejected is H_o: *A tutor-based approach for software visualization tool does not significantly improve the quality of software written by novice software engineers.* The proposed TubVis approach is integrated with existing reverse engineering environment (UDARE) that has been developed by the researchers. UDARE is currently under the process of finalising the implementation and integration with the enhancement of researchers' existing method of software visualization [20][21] in SoVis tool.

UDARE allows a software system to be inputted and then software artefacts will be extracted (refer Figure 2). Based on the extracted artefacts, the proposed TubVis will analyse software dependencies and state some recommendations based on the input made by the experts. The experts here can be project leaders or software managers who guide novice software engineers. Besides for computer science students, their lecturers can be the experts who will determine the parameters to be checked in the project assigned to them.

Then a view that uses graphical notations will be generated together with the recommendations. The tool will indicate 'why' the design is not recommended. For instance a circular class dependencies is not a good design. TubVis will indicate why it is not a good design that is where the circularity occurs and suggests how to break the circularity. Another important attribute of software design is coupling and cohesion.

As we know, a good design should have high cohesion and low coupling. These criteria are checked using fan-in (number of classes that call a particular class) and fan-out (number of classes that are called by a particular class). A lot software projects such as [13] uses fan-in and fan-out as one of the metrics to evaluate their projects. Fan-in and fan-out can be measured either at class level of method level. High number of fan-in indirectly shows the class or the method promotes reuse. High number of fan-in or fan-out among methods in a particular class reflects high cohesion, which is a good design. On the other hand, high number of fan-in or fan-out among different classes will lead high coupling that should be limited to certain reasonable numbers. This is the reason why expert software engineers who understand the overall project requirements should set the parameters related to the concerned attributes of software design in this case modularity.

Thus TubVis will be able to detect any un-recommended features as defined by the experts to be highlighted to the users. The tutor-based approach will indicate best practices in software design hence it indirectly promotes better coding of software systems. The focus of this research limited to providing guidance or tutoring to novice software engineers involving the transformation of views in design stage to source codes and vice versa. Other stages of software development or maintenance will be considered in the future work. The details of TubVis will be explained in the following paragraph.

Let S is the source code parsed by UDARE that output a set of X artefacts. X consists of $x_1 \ldots x_n$ that

978-1-4244-2328-6/08/$25.00©2008 IEEE

can be packages, classes, methods or data. Let *R* is a set of rules archived by software engineering experts for specific stages of software development. Upon users' selections on type of checking required, TubVis will make an analysis of the relevant artefacts of set *X* in order to generate a tutor guide or experts' recommendations. Finally the graphical view generated is accompanied with a set of recommendations or guidance *G*. Software engineers need to decide whether to change their design as required or they just proceed. If they proceed, the "defects" will be further accumulated in the next stage of checking in order to highlight their design or coding deficiencies. In the proposed approach we provide the flexibility by allowing the experts to either enforce the rules or not towards the novice software engineers. If the experts enforce the set of rules *R*, then the design needs to be corrected before software engineers can resume with writing source codes.

4. Implementation of TubVis approach in SoVis tool: an example

This section will give an example of how TubVis approach will generate tutor guides to a particular source code and design after UDARE has parsed and generate the view via a visualization tool called SoVis. This tool was developed using the DocLike Modularized Graph (DMG) method of the main researcher [20]. There are three main modules in TubVis approach: (i) Checker module, (ii) Graph module, and (iii) Documenter module. In this example we tested on a sample project that consists of 2 packages *CarRental* and *Calculator* as listed in Figure 3.

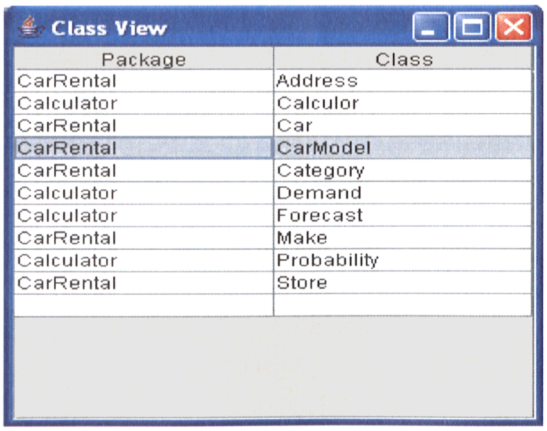

Figure 3: Class view provides the list of classes in all packages of a software project

978-1-4244-2328-6/08/$25.00©2008 IEEE

4.1 Checker module

Prior to checking the design, software engineers or users need to feed in the source codes into UDARE parser (see Figure 2). Then the experts need to define what are the criteria need to be set and to be followed by the fellow project members. The setting is done via the checker module as in Figure 4. In practice, project leaders or software managers should know the size and the scope of the project while planning the schedule of the project. The parameters entered into the checker module are considered a set of recommendations or rules given by the experts to the novice software engineers.

After the recommendations have been set, both project leaders and the fellow project members can check their source codes parsed to ensure they follow the rules set. If the project leaders set the enforcement to true value, each component will be checked whether it violates the recommendations or not. For instance the sample project should have only 5 classes per package (see Figure 4). If the number of classes that are checked on a particular package is more than 5, then it violates the rule. Thus the checker suggests the violation that requires novice software engineers to refine their design and source codes correspondingly as in Figure 5.

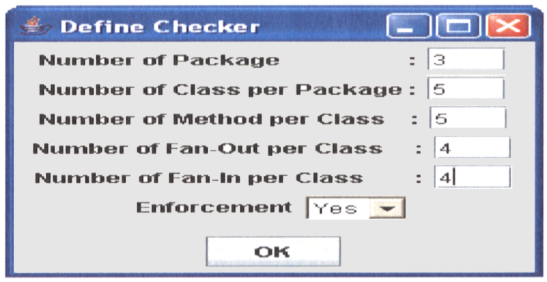

Figure 4: An interface to set the recommendations by the experts

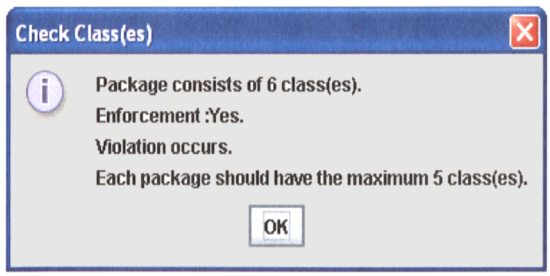

Figure 5: The checker module displays the result of the analysis that consists of a violation of rules set

If circularity occurs, the classes involved are recommended to be refined in order to avoid such design to be further accumulated in the software project. The example is shown in Figure 6 in which class *CarModel* is suggested not to call class *Store* to break the circularity. This will guide the novice software engineers to rectify their design before further continuing with the writing of source codes and repeating the same mistakes.

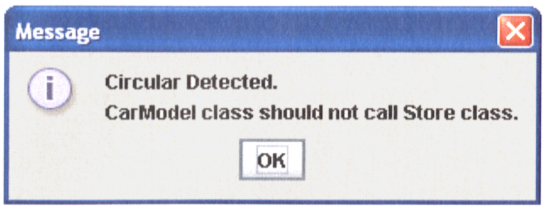

Figure 6: Circularity is detected and should be removed from the corresponding classes

4.2 Graph module

Alternatively, once the source codes have been parsed and the design has been checked via the Checker module, the artifacts can be visualized as in Figure 7. In this example the 6 modules are interconnected with each other without producing a circular relationship anymore as detected earlier in Figure 6. The number of fan-in and fan-out can also be viewed by right-clicking the concerned node.

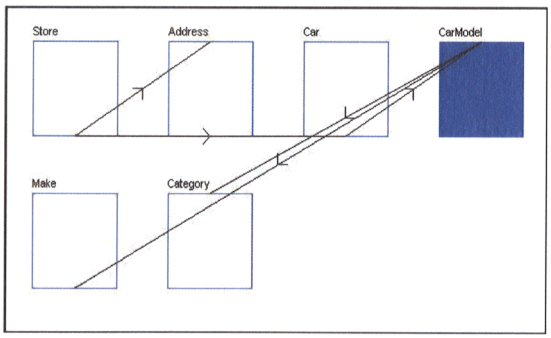

Figure 7: A graphical view shows dependencies among classes with the searched class highlighted

In order to give a flexible approach to the users, they may be allowed to proceed with the design that have violations of recommended practices. However the TubVis tool may be set to be inflexible by the experts in order to avoid novice software engineers to ignore the recommendations. Using this approach, users are forced to rectify their design before they proceed to the next stage. Hence this will ensure software engineering discipline is practiced at the very beginning of software development or maintenance.

4.3 Documenter module

The documenter module allows novice software engineers to set the required graphical views to be linked to the corresponding sections as in Figure 8. The semi-automation of document generation allows novice software engineers to focus on producing good design and source codes rather than documents. It cannot be denied that producing system document is vital, yet too much concentration on document writing will affect the project schedule. With TubVis approach we attempt to reduce the gap by having both Checker module and Documenter module to complement each other. The users are also able to use existing templates created or project leaders may create a custom template to be followed by their project team members. This will directly promotes better software engineering practice.

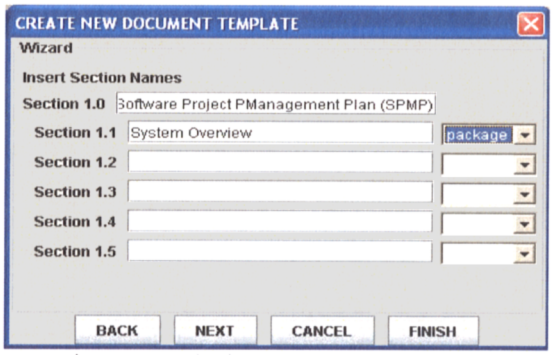

Figure 8: Each view can be set to link with corresponding sections in the document template

5. Related work

As we discussed earlier in the introduction, commercial CASE tools and IDEs provide a comprehensive environment for software engineers or computer science students to draw diagrams during analysis and design stage and then transform the design into corresponding source codes. Rational Rose [14] is an ubiquitous commercial tool that provides both forward and reverse engineering so-called roundtrip engineering. However the tool is very expensive causing the academic institutions or small software departments opt to employ open source tools available or do not even use such tools at all. In this case novice software engineers will use drawing tool to design their software and then

978-1-4244-2328-6/08/$25.00©2008 IEEE

transform manually into source codes using any available IDEs for the software languages they use. For instance they might draw a class diagram using Microsoft Word and then implement the coding using Borland JBuilder to develop a Java-based software system.

In addition, commercial tools like Rational Rose [14] are quite complicated to be used by novice software engineers particularly students. Such tools are more appropriate to be used by experienced or expert software engineers if they are fully equipped with best practices in software engineering in particular software design and coding. Users of the tools will be able to generate diagrams such as use case diagram for analysis and then create the sequence diagram during the design stage. The tool allows generation of source codes' skeleton from the sequence diagram. However the tool does not check whether the diagrams in both analysis and design stage are correct or conform to software engineering discipline or best practices. They do not provide direct guidance or tutoring tool to suggest to novice software engineers or computer science students how to analyze, design and write source codes that conform to the best practices in order to produce high quality software.

Hence this has motivated us to integrate tutor-based approach to visualise software systems in a reverse engineering environment and then the extracted software artefacts will be analysed by comparing them with the analysis, design and coding rules. A set of recommendations will be generated once the rules have been checked. For instance, the rule of cohesion and coupling of classes can be archived in the database and then the rule is checked when analysing extracted artefacts consist of a number of interrelated classes.

Referring to the main research question indicated earlier: *How to produce a software visualization tool that can provide both practical and theoretical guidance in software engineering while designing and coding software systems?* The main issues to be considered include what aspects to be considered in order to provide a software visualization tool that not only generate graphical views and provide the on-line help or user manual on how to use the tool but it should also provide a theoretical guidance to software engineers particularly novices such as students. This problem partly has been pondered by some researches focusing on the ability of students to understand object-oriented software [8][16][17], comprehend programs using animation [11][12] and understand software for maintenance [1][18][23].

Jimenez-Diaz *et al.* [8] proposed the use of virtual role-play by computer science students in a virtual 3D environment. Role-play is claimed to be an active learning approach in which the tool indicates the behaviour performed by the role-play including name, goal, actors and description. During the simulation, the students can observe the participating objects, their run-time classes, active objects, and their scope, control flow and method calls. This approach targets to assist understanding of object-oriented software system but it does not provide any recommendation or guidance whether the software written corresponds to the design and analyse the correctness of the design.

Shull *et al.* [16] reported how an object-oriented framework employing example-based techniques are more suitable to beginners compared to hierarchy-based techniques, which the learning curve is quite huge. The work indicated that example-based technique that guide students to explore an example by understanding a particular object in the source code is more effective for beginners of the framework. On the other hand, hierarchy-based techniques, which guide students starting to understand high level or broad classes of functionality towards a deeper level of classes and specific instantiation, are not suitable for beginners to understand software artefacts in an object-oriented framework. This study shows the best reading technique to be employed by beginners. It does not suggest any tutoring element to improve the novice's understanding in designing and coding object-oriented software.

The work of McWhirter [11] suggested the use of an animation system to assist students to understand behaviour of programs. For instance in order to aid students' understanding regarding a breadth first search algorithm, the tool called AlgorithmExplorer produces a graph-based animation that allows students to input values to the program and observe the changes via the animated graph. Another work of Ohki and Hosaka [12] promoted PAVI (Program Action Visualization Interpreter) that interprets and visualises program behaviour. The tool represents variables, arrays, and pointers as three-dimensional objects. Then it animates the actions of an object when there is an assignment operation. Both AlgorithmExplorer [11] and PAVI [12] focus on the understanding of source codes via animation without any checking element for the design and coding best practices.

More related work like that of Lowry [10] has proposed a computer-based tutorial resource to support students to understand complex commercial

978-1-4244-2328-6/08/$25.00©2008 IEEE

CASE tools in order to demonstrate software engineering concept. Wood and Danielson [24] suggest a web-based tutorial resource for introductory logic design course that enable students to learn the available topics and then discuss and review text interactively. Hacker and Sitte [7] developed WinLogicLab teaching suite that not only provide materials of digital logic design course but also allow students to interactively produce the design. However such work target more on educational aspects which differ from our proposed work that integrates both theoretical and practical aspects of software engineering into a CASE tool using both reverse engineering and software visualisation method in order to guide novice software engineers or computer science students.

Based on the literature study conducted so far, none of existing work has explicitly proposed a tutor-based approach as the guidance for novices while developing software system using a CASE tool.

6. Conclusion and future work

This research paper has proposed TubVis, a tutor-based software visualization approach that is integrated with a software visualization tool (SoVis) in a reverse engineering environment. The software artifacts are extracted and visualized together with the recommendations of how good the artifacts produced compared with the rules or best practices archived by software engineering experts. The approach focuses on novices or beginners in software engineering including computer science students who need both theoretical and practical guidance while developing a software system specifically in design and coding stages. Hence the integration of the proposed approach in CASE workbenches such as software visualization tool is anticipated to be able to improve quality of software designed and developed. TubVis also promotes better understanding of written source codes by novices. Thus the end software product will probably have better quality if they are developed and maintained according to software engineering principles.

Our future work will be to further expand the proposed approach to other stages such as analysis and testing. Current work focuses on design and coding stage only. The checker module will expand to be applied in a forward engineering tool. Then we will further evaluate the significance of the proposed approach in system development among novice software engineers to reject the null hypothesis stated at the earlier part of this paper.

978-1-4244-2328-6/08/$25.00©2008 IEEE

7. Acknowledgements

This work is supported by the Ministry of Science, Technology and Innovation (MOSTI) Malaysia: e-Science fund 305/PKOMP/613119. We would like to also acknowledge SoVis and TubVis research group members: Md Yamin Md Yusof, Shahrul Ismail and Intan Juhaina Johan.

8. References

[1] Austin. M. A., III, and Samadzadeh, M. H., "Software Comprehension/Maintenance: an Introductory Course", *18th International Conference on Systems Engineering, ICSEng 2005*, 2005, pp. 414-419.

[2] Borland, *Borland IDE: JBuilder 2007*, http://www.borland.com/de/products/jbuilder/, 2007.

[3] Buchsbaum, A., Chen, Y-F., Huang, H., Koutsofios, E., Mocenigo, J., Rogers, A., Jankowsky, M. and Mancoridis, S., "Visualizing and Analyzing Software Infrastructures", *IEEE Software*, September/October 2001, pp. 62-70.

[4] Budgen, D., *Software Design, Second Edition*, Pearson Education, UK, 2003.

[5] Chikofsky, E. J. and Cross II, J. H., "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, January 1990, pp. 13-17.

[6] Din, J. Ali, N. M., Mohd Noah, S. A. and Idris, S., "A Conceptual Framework of Object-Oriented Design for Novice Designers", *The First Malaysian Software Engineering Conference (MySEC'05)*, USM, Malaysia, 2005, pp. 83-88.

[7] Hacker, C. and Sitte, R., "Interactive Teaching of Elementary Digital Logic Design with WinLogiLab", *IEEE Transactions on Education*. Vol. 47, Issue 2, May 2004, pp. 196-203.

[8] Jimenez-Diaz, G., Gomez-Albarran, M. Gomez-Martin, M. A. and Gonzalez-Calero, P. A., "Understanding Object-Oriented Software through Virtual Role-Play", *IEEE International Conference on Advanced Learning Technologies, ICALT 2005*, pp. 875-877.

[9] Lanza, M., "Lessons Learned in Building a Software Visualization Tool", *Proceedings of the 7th European Conference On Software Maintenance and Reengineering (CSMR'03)*, 2003, pp. 1-10.

[10] Lowry, G. R., "CAL Support for Complex CASE Tutorials. Demonstrating Software Engineering Concepts through CASE", *International Conference on Software Engineering: Education and Practice, IEEE* Computer Society Press, USA, 1996, pp. 292-299.

[11] McWhirter, J. D., "AlgorithmExplorer: a Student Centered Algorithm Animation System", *IEEE Symposium on Visual Languages*, 1996, pp. 174-181.

[12] Ohki, M. and Hosaka, Y., "A Program Visualization Tool for Program Comprehension", *IEEE Symposium on Human Centric Computing Languages and Environments*, 2003, pp. 263-265.

[13] Project Metrics Help – Structural Fan-In and Fan-Out Metrics, http://www.aivosto.com/project/help/ pm-sf.html, 2007.

[14] Rational, *IBM Rational Software*, http://www-306.ibm.com/software/rational/, 2007.

[15] Rigi, *Rigi Group Home Page,* http://www.rigi.csc.uvic.ca/, 2007.

[16] Shull, F., Lanubile, F. and Basili, V. R., "Investigating Reading Techniques for Object-Oriented Framework Learning", *IEEE Transactions on Software Engineering*, 26(11) Nov. 2000, pp. 1101-1118.

[17] Soga, M., Kashihari, A. and Toyoda, J. –I., "Designing a Self-Explanation Environment for Multilayer Understanding. In Case of Program Understanding", *IEEE International Conference on Multi Media Engineering Education*, 1996, pp. 49-57.

[18] Stimick, J., "An Undergraduate Course in Software Maintenance and Enhancement", *Tenth Conference on Software Engineering Education & Training*, 1997, pp. 61-73.

[19] Storey, M. –A. D., Fracchia, F. D. and Muller, H. A., "Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration", *The Journal of Systems and Software,* 1999.

[20] Sulaiman, S., Idris, N. B. and Sahibuddin, S., "Enhancing Cognitive Aspects of Software Visualization Using DocLike Modularized Graph (DMG)", *International Arab Journal of Information Technology (IAJIT)*, 2(1), Zarka Private University, Jordan, 2005, pp. 1-9.

[21] Sulaiman, S., Idris, N. B., Sahibuddin, S. and Sulaiman, S., "Re-documenting, Visualizing and Understanding Software Systems Using DocLike Viewer", *10th Asia Pacific Software Engineering Conference*, IEEE Computer Society Press, USA, 2003, pp. 154-163.

[22] Sulaiman, S. and Sulaiman, S., "A Tutor-Based Software Visualization Approach (TubVis) for Novice Software Engineers", The Second Malaysian Software Engineering Conference (MySEC'06), UTM Press, Malaysia, 2006.

[23] van Deursem, A., Favre, J. –M. Koschke, R. and Rilling, J., "Experiences in Teaching Software Evolution and Program Comprehension", *11th IEEE International Workshop on Program Comprehension*, 2003, pp. 283-284.

[24] Wood, S. and Danielson, R., "Java-Based Instructional Materials for Introductory Logic Design Courses", *30th Annual Frontiers in Education Conference 2000*, S2D/10-S2D/16, Vol.2, 2000.

978-1-4244-2328-6/08/$25.00©2008 IEEE